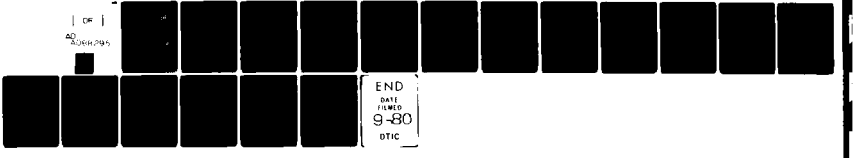
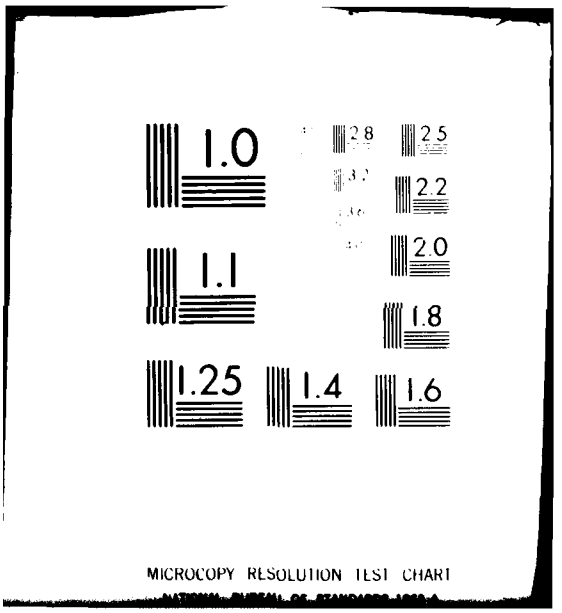
AD-A088 295 CARNEGIE-MELLON UNIV PITTSBURGH PA MELLON INST OF SCIENCE F/G 9/2
BMD RECONFIGURABLE TOPOLOGY STUDY, DDP EMULATION STUDY.(U)
FEB 80 R L KRUTZ DAS660-79-C-0043
UNCLASSIFIED NL



END
DATE FILMED
9-80
DTIC


1.0
2.8
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART

MELLON INSTITUTE

COMPUTER ENGINEERING CENTER

(2)

LEVEL II

FINAL REPORT 3779 - 2/80

BMD Reconfigurable Topology Study

DDP Emulation Study

Contract No. DASG60-79-C-0043

Submitted By

Dr. Ronald L. Krutz

SPONSORED BY

THE BALLISTIC MISSILE DEFENSE ADVANCED TECHNOLOGY CENTER

CONTENTS:

1. Preface

This document serves as the final report for work performed for the Ballistic Missile Defense Advanced Technology Center, Huntsville, Alabama, under Contract Number DASG60-79-C-0043, during the period March 1, 1979 to February 29, 1980. This document should be considered to include information concerning the work performed during the last month of the contract period, as well as the summary report for the 4th quarter of the contract period.

2. Introduction

During the contract period, work has been focused in five major areas:

1) Development of the BUCS-1 system hardware and software.

2) Development of logic analysis procedures for DDP systems.

3) Definition of the concept of "resource emulation".

4) Definition of processing system requirements for potentially BMD related signal/image porcessing tasks.

5) Definition of possible research tasks within the framework of the requirements defined in 4), using BUCS-1 or a similar system as a research tool.

A summary of the major points of investigation and the work performed for these areas follows.

3. BUCS-1 System Development

During this contract period significant work was performed on the BUCS-1 system both in the area of hardware implementation and in the area of software development. This included such tasks as an analysis of the potential contention problems associated with the operation of the interprocessor interrupt bus followed by the design and implementation of a contention free bus, and the design and initial development of HLL (high level language) run time library routines. The major devlopments are summarized below.

3.1. Interrupt Bus Contention Analysis

The original interprocessor interrupt bus design relied entirely upon software (which presumably was to have been included in the "operating system" associated with each microcomputer) to control the operation of the bus and to resolve access conflicts. This would have proved difficult to implement as the Z80 microprocessor instruction set does not contain a "Test and Set" instruction, and the hardware design of the bus would not have supported any of the other schemes which can be used for process synchronization.

Therefore, a decision was made to redesign the hardware to support discrete time-slice access by the individual microcomputers to the interprocessor interrupt bus. In this system, bus access grant is "passed" from processor to processor in round-robin sequence. If a program requires access to the

interrupt bus, a write operation is performed to an output port with the destination processsor address and interrupt-type code. A request bit is then set in another port, and when the "bus access grant bit" arrives, an interrupt bus cycle is performed before passing the grant to the next processor. A status bit is set when the bus cycle is completed, indicating to the software when the interrupt bus interface may next be accessed.

This bus system, although theoretically correct, has proven to be difficult to debug, and occasionally malfunctions due to transient error conditions. Future implementations of interprocessor interrupt capability would attempt to design simpler, more reliable hardware, that would provide more capabilties than the present system provides.

3.2. System Hardware Integration

The individual microcomputers that comprise the BUCS-1 system underwent final assembly and debugging during the contract period. This work took place over a period of approximately three months.

Integration of the individual microcomputers into the whole system proved to be somewhat more difficult than debugging the microcomputers. Difficulties were encountered with intermittent connections on the intercomputer buses, interface chip failures, and a design error in the bus repeater circuitry which caused all bus lines to latch in the asserted state. In addition, the operating system routines controlling intercomputer communication were being developed at the same time as system integration was proceeding, making it difficult to distinguish between software

errors and hardware failures.

The bus repeater circuitry was permanently disabled after analysis of the circuit showed that the information necessary to control the repeater was not available in any combination of the control signals on the buses. However, the visual bus monitor feature of the repeater box was retained and proved to be a valuable debugging tool.

System integration was successfully completed in May 1979, in time for a demonstration to the DPAD working conference held at MI-CEC.

3.3. Monitor Software Development

During this contract period, the "operating system" or stand alone monitor for the individual microcomputers of the BUCS-1 system was designed, coded, and debugged. The monitor software provides the standard features of an emulated "lights and switches" front panel (examine memory location, deposit to memory location, initiate program execution from a specific address), as well as several user convenience feaures (block move of memory, block fill of memory, optional verify on memory deposit), and serial line download capabilities. In addition, the monitor provides support for interprocessor data transfer with data bus read and write, and interprocessor interrupt routines. These routines are available for use by the user's application program, but are provided principally for downloading code to the various microcomputers of the BUCS-1 system from MI-CEC's host computer.

3.4. HLL Support Development

As part of the overall BUCS-1 system implementation, high level language programming support was deemed to be a desirable feature. Since the MI-CEC host computer supporting BUCS-1 runs the UNIX operating system, which supports the 'C' programming language, a 'C' compiler producing Z80 native code was obtained for internal use in supporting BUCS-1. The 'C' programming language proves to be a good compromise for programming microcomputers as it is close enough to the hardware to permit the programmer to manipulate hardware features, yet the language supports high level structured programming concepts.

To provide an interface between the microcomputers and the 'ZC' (Z80 'C') compiler, certain run time routines had to be developed. These included the terminal support routines "cttywt" and "cttyrd," the interprocessor data transfer routines "cbuswt" and "cbusrd," and the interprocessor interrupt routines "cprcin" and "cinterrupt." These routines were successfully coded and a test program using these routines was run on multiple processors.

Future development of 'C' support would include interfacing to the interval timer hardware, and the coding of calls to the floating point software package.

3.5. Demonstration Application Software

As part of the overall BUCS-1 system integration procedure, and to demonstrate multiprocessor capability, a simple bubble sort test algorithm was implemented and run on three different system configurations. Timing measurements were made of the algorithm runtimes using both hand held stopwatches and the

onboard interval timers. (Timing results were consistant between the two methods of timing.)

The data base that was sorted for all configurations consisted of a 252 byte array of random integers that had been generated by the host computer. The three configurations tested were uniprocessor, 3 processor ring, and a 3 processor tree. For the 3 processor ring, the sort was implemented by having each processor make a single sorting pass through the data array and then passing it on to the next processor in the ring. This continued until the array had been completely sorted, at which time it was passed back to the "master" processor (the one communicating with the host). For the 3 processor tree configuration, the "master" processor split the data base into three unequal sized parts and transmitted two of the parts to the "slave" processors. All processors sorted their data, and then the "slaves" sent their sorted subarrays back to the "master" for a merge.

The results of this exercise followed expectations: the ring configuration was limited by data transmission time as the 252 byte arrary was passed around the ring; the "divide and conquer" method of the tree configuration was the fastest because of the smaller arrays to be sorted and transmitted; and the uniprocessor was "average." The measured times for this experiment are:

Uniprocessor: 3.3 sec.

Ring Config: 58.9 sec.

Tree Config: 1.1 sec.

From measurements made with a logic analyzer attached to one of the data transfer buses, the average data transfer rate, using a fully handshaked transfer protocol, was determined to be 1.04 kilobytes/sec.

This experiment proved the multiprocessor capabilities of the BUCS-1 system, and more importantly, showed the limitations of software controlled bus protocol. Any future system designs, or any modifications made to this system, will attempt to place in hardware more of the data transfer control mechanisms.

4. Logic Analysis of DDP Systems

One of the problems encounterd while debugging and testing the operation of the BUCS-1 system was that of effectively using commercially available logic analyzers to monitor the performance of individual microcomputers and of multiple processor systems. As a result of these difficulties, an investigation into logic analysis techniques for DDP systems and the required logic state analyzer capabilities was begun. This investigation culminated in the development of a procedure to debug and "bring up" multiple (micro)processor systems, and a definition of the logic state analyzer capabilities required to support performance analysis of DDP systems and to implement the concept of "directed data storage," which was outlined as a user interface

improvement.

4.1. DDP System Logic Analysis

A 7 step logic analysis procedure was developed for dealing with multiple and/or distributed processor systems. This procedure, predicated on the use of a logic state analyzer with additional diagnostic information provided by oscilloscope and diagnostic software, calls for bringing up the system in the following order: nodal hardware, internodal hardware, local monitor software, local application software, and internodal application software. The last two steps of the procedure provide for verification of application software correctness.

4.2. Performance Analysis Techniques

Performance analysis of distributed or multiple computer systems by logic analyzers must be based on information available external to the system nodes. Thus, internodal signals are considered observable while intranodal signals are not. These observable signals, coupled with information concerning the application software running on the system, can be manipulated by a properly designed logic analyzer to provide meaningful displays of data to the user, who must then decide whether the given application software running on the given hardware configuration is performing as desired. The types of displays that would provide enough information for the user to make an informed decision include such things as: displays of "source-destination" pairs in a "map" format, "map" displays of qualified intranodal operations (reads, writes, transfers of specified data, accesses to specified nodes, etc.), tabular displays of

data transfers or node acesses, and displays of relative nodal activity based upon a user defined activity parameter.

4.3. Logic Analyzer User Interface

A major result of this investigation was the determination of the need for better user interfaces to the logic analyzers. One specific user interface improvement would be the implementation of "directed data storage." In this technique, the "category" of a sampled state is determined from other previously, or simultaneously, monitored information. The sampled state is then stored under the correct "category" in the logic analyzer, and the display of traced states is formatted to show relative time relationships between states in the various categories. This technique would enhance the observability of various system conditions. It also allows the user to attach meaningful labels to groups of states, which labels are displayed with the associated states.

This, and similar improvements in the user interface would enhance the task of performance analysis discussed above.

5. Resource Emulation Concept Development

After construction and testing of the BUCS-1 system, a search was begun for a task (or tasks) related to BMD ATC's area(s) of interest that could be successfully investigated with BUCS-1. Two avenues were investigated: that of using the BUCS-1 system as a programmable "resource" emulator, and using BUCS-1 as an emulator of part of a signal/image processing system. This latter task included an analysis of processing system requirements for signal/image processing tasks, followed by the

definition of a possible task for a BUCS-1-like system. These are described in later sections.

5.1. Resource Emulation - Definition

Under the concept of "resource emulation," the individual microcomputers of the BUCS-1 system would be programmed to simulate the operation of PMS level primitives. These primitives (processor(s), memories, I/O devices, etc.) would communicate via the BUCS-1 data transfer buses using emulated bus protocols, such as Intel Multibus, S-100, DEC Unibus, etc. Thus, BUCS-1 would be emulating a proposed system.

5.2. Example of Memory Emulation

Part of the research into the use of BUCS-1 as a "resource emulator" investigated the question of the range of emulation slowdown factors that could be expected. Specifically, the effect of optimizing the simulation code was investigated. Two different simulations of a 4 kilobyte static RAM with a 250 nanosecond access time were written. The first was a well structured, well organized program, while the second was hand optimized for speed of execution. Instruction execution times were calculated for both coding styles (assuming a 2 megahertz clock driving the simulating processor) which yielded a slowdown factor of 388:1 for the "structured" code, and a slowdown factor of 225:1 for the "optimized" code, for a ratio of 1.72. This amply demonstrates the importance of optimizing the simulation code in order to obtain faster emulations.

6. Signal Processing System Requirements

As mentioned in the previous section, one avenue of research included an analysis of processing system requirements for signal/image processing tasks, as it was felt that the emulation of signal/image processing nodes might be a suitable task for BUCS-1. This investigation was roughly divided into the following two areas, which will be outlined below: an analysis of control flow processor requirements, and an analysis of data operation primitives.

6.1. Control Processor Requirements

One valid approach to the structured design of a distributed system is to decompose the system design requirements into data flows and control flows. One conclusion that can be drawn from this is that distributed BMD signal/image processing systems could be designed using two classes of devices, one for data operations and one for control functions.

The "control processor" design requirements can be summarized as requiring a high degree of flexibility, implying a certain amount of programmability. Specific requirements include:

- Ability to communicate with other devices
- Ability to control multiple tasks
- Ability to work with compatible controllers in a DDP environment
- Ease of programming and reconfiguration

- Software compatibility with other systems

Certain of these requirements imply that a control processor's architecture should be tailored to a "universal" high level language that supports multi-tasking, such as Ada. Thus, an "Ada processor" acting as a control element would manage data flows and communicate with other controllers and data operators in a typical BMD signal processing system.

6.2. Data Operator Primitives

An analysis of various signal/image processing algorithms used in diverse applications showed that there were many common data operation functions. This algorithm analysis, which included both time/space and frequency domain applications, yielded the following set of functions common to many of the applications:

Auto correlation

Cross correlation

Power spectrum

Histogram

FFT and Inverse FFT

Convolution Filter

Recursive Filter

Statistical measurement of samples

Specific algorithms performing tasks in noise reduction,

signal enhancement and restoration, signal registration, and image segmentation and recognition

Data and sample acquisition

Display control

These functions were further analyzed to determine the most common computational level primitives that are required to implement these functions. This analysis showed that one of the most pervasive computational structures is that of an array complex add. A need for the following primitives was identified:

Real and complex vectore arithemetic: add, subtract, multiply, divide, absolute value, etc.

Real and complex matrix arithmetic: add, subtract, multiply, inverse, eigenvalue, eigenvectors

Real and complex vector multiply-add

Coordinate system conversion

Data format conversion

Packing and unpacking of data structures

Vector integration and differentiation

Differentiation and integration of two-dimensional data

One conclusion that can be drawn from this brief investigation is that these computational primitives could serve as set of language primitives in a high level signal processing

language, which would among other attributes simplify software development for signal/image processing systems, expand the number of users for such software, and standardize notation and documentation in this area of software development.

7. Proposed DDP Tasks for BUCS-1

Based on the analysis of control processor requirements and data operator primitives, a possible research task, related to potential BMD signal processing requirements, using BUCS-1 (or a BUCS-1-like system) was defined.

In this task, the BUCS-1 system would be front ended and controlled by a microcodable LSI-11 which would emulate (a subset of) the actions of a high level language control processor. This configuration would permit the user to test various high level language control structure primitives, and the mapping of higher level signal or image processing primitives onto hardware units, in this case, the BUCS-1 processors, which could be considered as hardware implementations of various processing tasks (processes).

Based on the results of this experimental task, recommendations concerning the BUCS-1/LSI-11 system (hardware and software) would be made. These would cover such problems as the user interface, system reliability, and hardware changes to be included in future generations of multiprocessor emulator systems. Future directions of research would also be enumerated.

8. Conclusions

The R & D effort during the contract period has provided some fundamental insight into the development, debugging,

analysis, and programming of a distributed microcomputer testbed. The concepts which were verified, the real-time software that was developed, and the experiments that were run provide a basis for future work in high level language relationships to DDP microcomputer system architecture and high level implementation of system control. These factors are particularly important since the next generation of DDP microcomputer systems may well be implemented on wafers with reconfigurable capabilities which must be controlled by high level mechanisms.